Technical Report 1432
May 1991

High Speed Systolic Array Processor (HiSSAP) System Development Synopsis

Lesson Learned

J. P. Loughlin

NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commander

R. T. SHEARER, Acting
Technical Director

ADMINISTRATIVE INFORMATION

The study covered in this report was conducted from October 1983 to October 1990. It was funded by the Naval Sea Systems Command (NAVSEA), Washington, DC 20362. The project number is 76-EE3401, agency accession number DN308 022, and program element NIF, 604507N. The work was performed by the Processing Research and Development Branch (Code 761) of the Naval Ocean Systems Center, San Diego, CA 92152-5000.

Released by
G. W. Byram, Head
Processing Research and
Development Branch

Under authority of
F. M. Tirpak, Sr., Head (Acting)
Space and Systems Technology
Division

ACKNOWLEDGMENT

The High Speed Systolic Array Processor (HiSSAP) project encompasses work performed over a 7-year period at NOSC. The goal of this project was to obtain a clearer understanding of the complex interactions between parallel processing architectures and adaptive matrix-based signal processing algorithms. Sponsorship of system hardware and software development was initially obtained from the Lasers and Microelectronics NOSC program block managed by Dr. Isaac Lagnado, and the NOSC Research and Technology Branch (Code 553). During the intermediate phase of the project, mapping of the multiple-signal classification (MUSIC) algorithm and the finite impulse response (FIR) filter onto the testbed hardware was supported by joint sponsorship of the block and major bid and proposal discretionary funding (coordinated by Dr. John Silva, NOSC Office of Technology Research). Integration of these system software applications and the balance of the signal processing/data acquisition and the demonstration of the high-frequency direction finding application was obtained from the NAVSEA standard matrix processor project (PMS-412), with additional funds supplied by the Lasers and Microelectronics block.

LH

CONTENTS

FIGURES

SUMMARY

OBJECTIVE

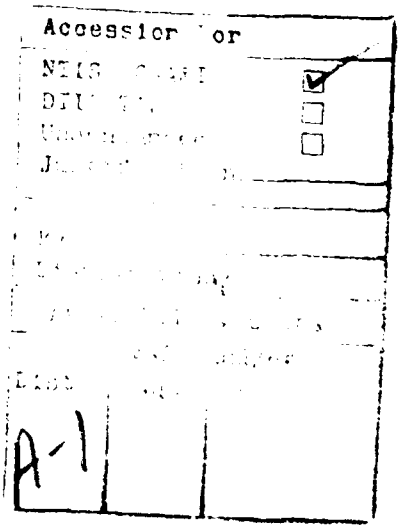
Document the design rationale of the High Speed Systolic Array Processor (HiSSAP) testbed.

RESULTS

In addition to reviewing general parallel processing topics, the impact of the HiSSAP testbed architecture on the top-level design of the diagnostic and software mapping tools is described. Based on the experience gained in the mapping of matrix-based algorithms on the testbed hardware, specific recommendations are presented in the form of "lessons learned," which are intended to offer guidance in the development of future Navy signal processing systems.

CONCLUSIONS

The interaction of algorithm and architecture within a parallel processing host is complex. The topics described in this document serve mainly to identify major architectural design issues that must be addressed if optimum system performance is sought. Once the specific requirements of the hosted algorithm (or class of algorithms) are identified, a parallel processor architecture that uses its resources efficiently can be designed.



INTRODUCTION

This report documents the design rationale of the High Speed Systolic Array Processor testbed (HiSSAP). In addition to reviewing general parallel processing topics (appendix A), the impact of the HiSSAP testbed architecture on the top-level design of the diagnostic and software mapping tools is described. Based on the experience gained in the mapping of matrix-based algorithms on the testbed hardware, specific recommendations are presented in the form of "lessons learned," which are intended to offer guidance in the development of future Navy signal processing systems.

BACKGROUND

Modern signal processing algorithms that perform well in complex and congested signal environments generally rely on massive amounts of computation performed on large arrays of input data (references 1 and 2). Significant advantage in communication and surveillance can be achieved if these high-performance algorithms can be mapped onto suitable hosts. To achieve the throughput necessary to process large arrays of input data in real time, signal processing architectures optimized for matrix operations must be identified.

In 1978, Professor H. T. Kung, an Office of Naval Research (ONR) sponsored researcher at Carnegie-Mellon University, reported his discovery of a new type of signal processing architecture known as the systolic array (reference 3). A systolic processor is a regular array of identical computational cells with common control, local interconnects, synchronous timing, and homogeneous data flow. This structure allows modular parallelism with throughput directly proportional to the number of computational cells.

Since 1979, NOSC has investigated this new architecture as the possible host in Navy signal processing applications. Starting with the Systolic Array Processor (SAP) (reference 4), the world's first hardware implementation of a two-dimensional systolic processor, the design knowledge gained has provided the impetus and background to develop additional testbed systems capable of demonstrating the computational potential of systolic architectures for Navy processing needs. During the subsequent years, the investigations were expanded to include the development of the Systolic Linear Algebra Parallel Processor (SLAPP) (references 5 and 6), the Video Analysis Transputer Array (VATA) (reference 7), and the High Speed Systolic Array Processor (HiSSAP) testbed (references 8 and 9). Algorithm mapping is continuing on the VATA and near-term plans include mapping a variety of signal processing algorithms onto the Intel iWarp[1] parallel processing array.

The Navy wishes to transition this parallel processing technology to a broad base of application platforms. The plans are to extend the AN/UYS-2(V) Enhanced Modular

[1]iWarp is a trademark of Intel Corporation

Signal Processor (EMSP) to incorporate a new functional element type, the Matrix Processor (MP). Optimized to efficiently execute linear algebra operations linked to wide bandwidth input/output (I/O), these functional elements will greatly expand the operational capability of the EMSP. The planning and specification of the MP draw from the experience base gained in systolic array development at NOSC, particularly with the HiSSAP testbed.

SCOPE

The architecture of the HiSSAP testbed (described below) incorporated a rich set of programmable resources and served as a laboratory vehicle for determining processor characteristics necessary to efficiently host complex algorithms.

To fairly assess the performance and suitability of each resource, however, the testbed was integrated into a high-frequency direction finding (HFDF) application demonstration (reference 10). In addition to the testbed hardware, the integrated performance of mapping and diagnostic software, an antenna simulator, and data acquisition subsystems were carefully analyzed. This report will be limited to documenting aspects of parallel processing architectures as they pertain to overall system performance. The programming resources necessary to efficiently use the processor's architectural resources are discussed elsewhere (reference 11). It must be noted that the enhanced performance attributed to each architectural resource must be carefully weighted against the incremental complexity of the required programming tools.

A discussion of the system development methodology is included in this report to emphasize the importance of using computer-aided design and behavior modeling early in the design cycle. The design experience obtained from the HiSSAP HFDF demonstration effort forms the basis for the general parallel processor design guidelines presented below.

A general discussion of parallel processing topics is included in appendix A. The design topics contained in the appendix reflect the broader base of experience derived for SAP, HiSSAP, SLAP, and VATA development efforts.

HISSAP TESTBED SYSTEM OVERVIEW

Various design aspects in the development of the HiSSAP testbed system have been documented in the literature (reference 8) and are summarized here for the purposes of reader orientation. The intent of this summary is to introduce terminology unique to this system design and used in subsequent sections of this report. The architecture embodied in the HiSSAP testbed does not necessarily represent an optimum implementation of systolic technology but serves as a general-purpose host for laboratory investigations of parallel algorithm mapping. Most of the detail design of the testbed hardware was finalized 6 years ago, and many of the technology constraints affecting the original engineering of HiSSAP are not presently applicable. Although rapid strides in device technology may render any particular hardware implementation dated, the issues addressed in the lessons learned sections have general application to parallel processing technology.

SYSTEM TOP-LEVEL ARCHITECTURE

The HiSSAP testbed system, shown in figure 1, is composed of 16 arithmetic processing modules (APMs), four input/output modules (IOMs), a system control module (SCM), and two peripherally connected IBM–AT-class personal computers (PCs). One of these PCs (called the experiment controller) hosts the software development effort (references 9 and 11) and coordinates array activities during algorithm execution. The remaining PC is used to develop simulated data, support communications with the systolic array elements during algorithm execution, and support performance analysis. Each of the 16 APMs are connected via four 40-bit bidirectional parallel data channels to adjacent APMs or to a neighboring IOM on each of the boundaries of the 4×4 square array. Data communication to hardware external to the array occurs via four external bidirectional ports (labeled: top, bottom, right, and left). A 40-bit data channel, called the data circus, links the IOMs into a network capable of moving data around the periphery of the systolic array structure. Communication between the elements of the systolic array and the system control module is handled by a global channel called the array control bus (ACB). An external extension to this ACB provides the experiment controller PC access to the program states of all of the testbed internal elements.

PROCESSOR ELEMENT ARCHITECTURE

The APM and IOM elements of the array share a common design foundation; however, the APM incorporates the additional circuitry required to perform floating-point computations and extended modes of I/O port communication. One major feature of the internal architecture of the APM (figure 2) is the emphasis on high-communication

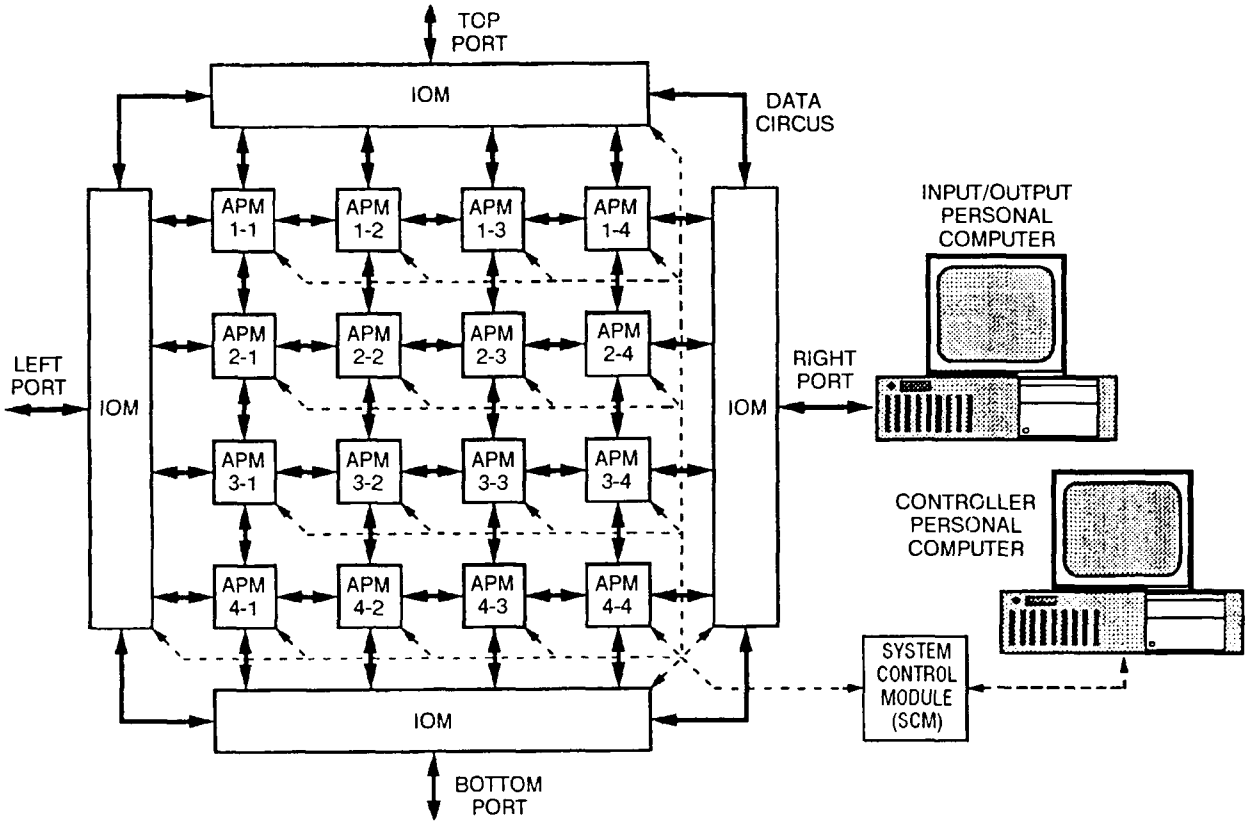


Figure 1. HiSSAP testbed system diagram.

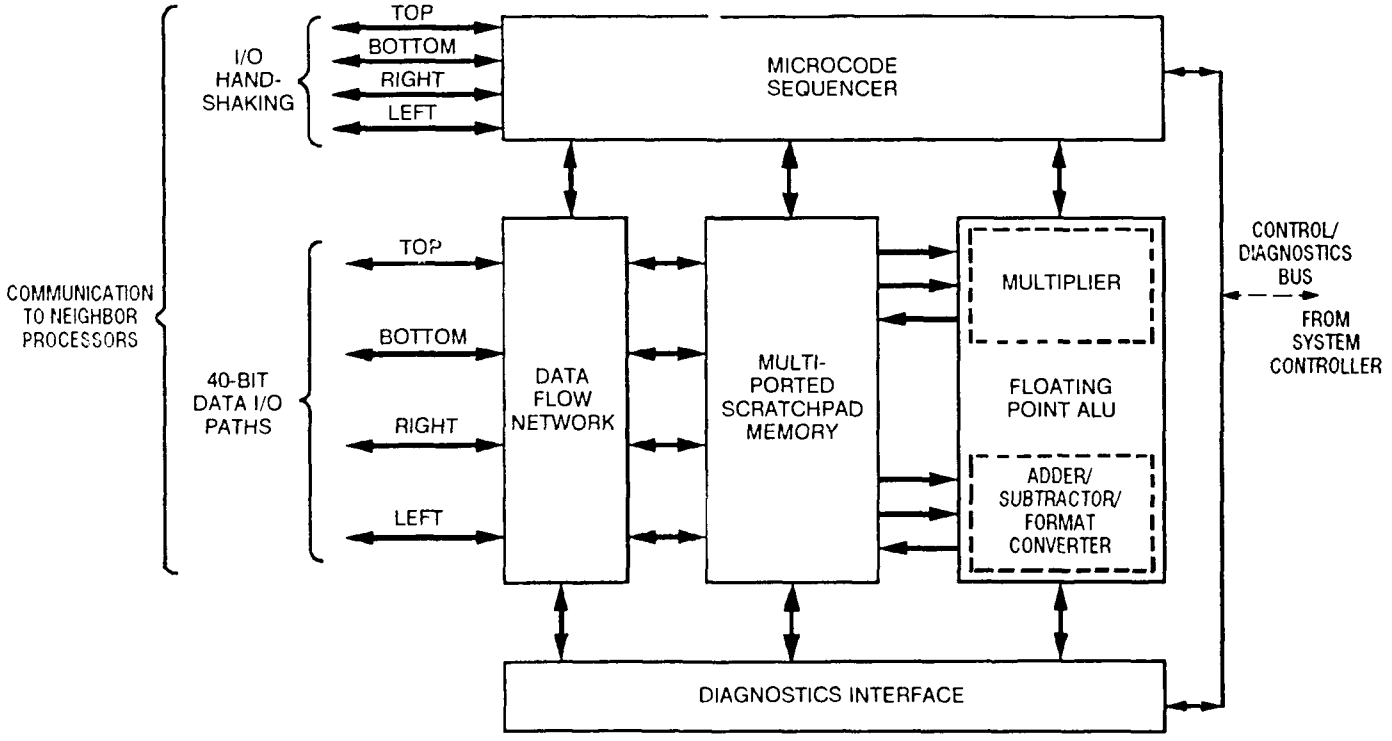


Figure 2. Arithmetic processing module (APM) functional diagram.

bandwidths of its resources and internal buses. The central processing unit (CPU) is composed of a pair of floating-point devices, one arithmetic logic unit (ALU), and one multiplier. These devices may be operated in parallel using separate access to a shared multiported memory. This memory also provides an interface to the four systolic data ports of the APM through a limited crossbar switching structure called the data flow network. The control of these element resources emanates from a 176-bit-wide instruction code sequencer capable of branching and nested loop execution. A network of set-scan test registers and an interface to the global ACB are included in the APM design to support diagnostic and control operations launched by the system controller PC.

The IOM design, similar to the APM in most respects, replaces the CPU function with a pair of additional data ports, the data circus, and the external I/O port (figure 3). Boundary I/O requirements allowed some economies of design to be applied to the multiported memory and data flow network in this module. The level of resource control required to operate this module type is substantially less, giving rise to a instruction word only 96-bits wide.

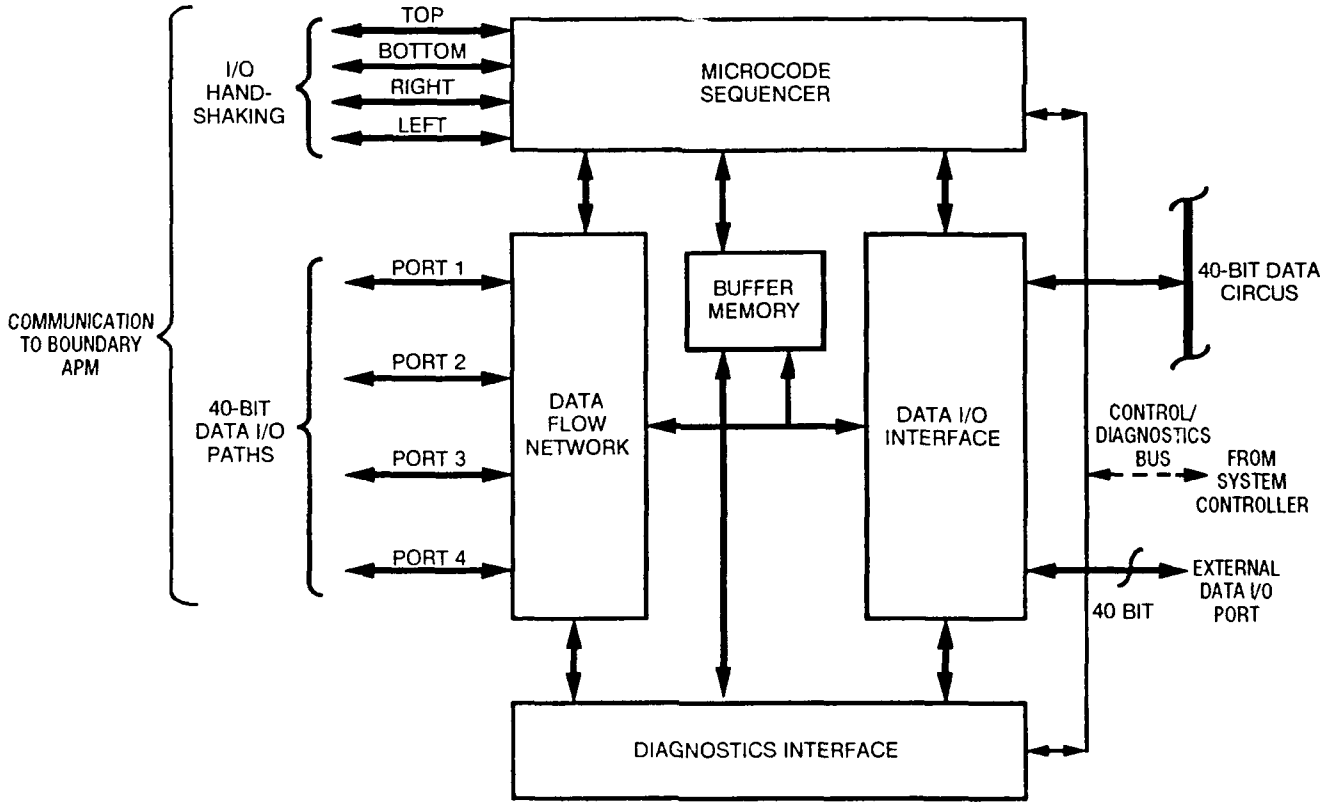


Figure 3. Input/output module (IOM) functional diagram.

HISSAP HARDWARE DESIGN TOOLS

The design of the HiSSAP processor system proceeded without the aid of either hardware or software simulation tools.

TESTING PHILOSOPHY

Design of the systolic testbed was started 7 years ago and finalized 18 months later. At that time, simulation tools were not available to adequately support such a design. In particular:

1. The processing modules within the array (while composed mainly of medium-scale integrated (MSI) digital logic devices) contained several large-scale integrated (LSI) and very large-scale integrated (VLSI) devices for which no suitable logic models had been generated. No behavior characterization libraries were available for such devices as the Weitek[2] 1033 Multiplier, 1032 ALU, and 1066 register file chips.
2. Characterization of device loading within the lengthy bus networks used in each processing module was not incorporated into design evaluation reports produced by existing simulation models. Due to the large board real estate, this loading could be further compounded by the mutual coupling of layered wire-wrap connections extending over several inches.
3. The behavior of the systolic backplane represented a challenging modeling task. Its physical size required that any model used for simulation would have to account for transmission line effects. The characteristic impedances of signal lines within this multilayer printed-circuit structure were difficult to estimate due in part to the proximity of several other signal layers and massive ground and power planes.
4. It was believed that due to the built-in set-scan shadow register design, the functional checkout of functional elements within each system module could be conducted in a straightforward manner. To minimize the debugging effort, a suite of diagnostic routines controlled by software executing on a personal computer was envisioned to exercise all the major functional elements and provide a means for localizing errors.

DEBUGGING EXPERIENCE

The debugging efforts of the HiSSAP testbed system hardware transitioned through three relatively distinct phases: (1) timing-related problems, (2) device I/O and signal problems, and (3) device behavior problems. Depending on the source of the functional problem, the level of diagnostic tools used varied.

[2] Floating-point devices manufactured by Weitek Corporation.

Timing-Related Problems

As expected, errors in logic design and documentation inconsistencies were discovered during various phases of initial checkout of the processor hardware. Many of the faults were traced to logic errors associated with the MSI circuitry and may have easily been avoided if only those sections of the design had been simulated using ordinary logic simulators. Marginal timing, simple gate loading, and logical inconsistencies were characteristic of this initial set of problems. This experience indicates that traditional design techniques are prone to human error and that as the system design grows, the time devoted to tracing down design faults of this type becomes prohibitive. Compared to the effort needed to isolate these design faults by traditional laboratory methods, the additional time invested in the construction and use of timing simulation models is justified. This is particularly important when major portions of system designs are cast-tered into monolithic form. As the technology of wafer-scale integration is transitioned to parallel processing architectures, it becomes mandatory to pursue timing models early in the design cycle.

Device I/O and Signal Problems

Another class of hardware fault, pathologic signal conditions or unspecified device interface behavior, required extensions to the testing methodology. Excessive signal ringing and improper clock transitions characterized this level of fault. It is unlikely that this type of fault would have been predicted by a simulator. Incorporating a variety of off-the-shelf integrated circuit components into the design of the HiSSAP testbed allowed great latitude in functional design and shortened the project development phase. However, the diversity in the methods with which each manufacturer documented device I/O performance contributed to the difficulty experienced in functionally debugging the system hardware. Difficulty in detecting and characterizing these faults during hardware system checkout identifies a need for system timing models to address the physical realities of the processor. Signal transmission line behavior and mutual coupling in clocking and busing circuits must be included in the timing simulation model in order to reliably predict system design integrity. Once quantitative estimates of critical system dynamics are available, the impact of architectural tradeoffs can be more realistically evaluated.

Device Behavior Problems

The final category of system debugging addresses the location of behavioral design errors. This effort sought to validate the designers understanding of manufacturer's specification of device function. The execution of programmable functions within a device (mainly LSI and VLSI) was verified using a combination of set-scan registers and microcode program fragments executed on the HiSSAP hardware.

This phase of system debugging was the most time-consuming and technically challenging. The internal registers and program states within the LSI and VLSI devices of each processing element were generally unaccessible for direct observation. This necessitated the creation and use of many specialized diagnostic software modules. In the absence of a suitable behavioral model, the functional verification of the diagnostic software could not be decoupled from that of the system hardware. The duration of the HiSSAP debugging effort, as a result, was greatly extended and required the extensive use of traditional laboratory test instruments. Undoubtedly, a major portion of this checkout effort could have been avoided if the initial design could have begun at the register description level. With the proper behavioral model, the functional integrity of the diagnostic modules used for the checkout of the system hardware could have been established prior to hardware construction. Further reductions in debugging time would certainly have resulted if the level of integration of the set-scan testing implementation could have extended into the functionality within the LSI and VLSI devices. For the development of the HiSSAP testbed, that would have resulted in a greatly reduced need for the diagnostic software modules written expressly for deducing the state of "hidden" registers within these monolithic devices of the architecture.

HISSAP SOFTWARE DESIGN TOOLS

The hardware debugging phase of the system development was supported by the use of custom diagnostic software developed on the personal computer HiSSAP controller. Three distinct levels of software tools were developed to aid functional testing of the processors: (1) board level, (2) array level, and (3) algorithm level.

BOARD-LEVEL TESTING

To perform the initial functional testing of the individual processor as a stand-alone element of the processor array, a family of low-level "primitive" software utilities was constructed. Using the set-scan diagnostic/control port of each board, these primitives provided a means of evaluating the functional integrity of each functional block within the processing element. Floating-point computations, program execution, and address generation were a few of the many functions tested by this level of test software.

SYSTEM INTEGRATION TESTING

The next level of diagnostic utilities addressed the operation of the processing elements linked in parallel test program execution. This category of testing required the greatest development effort due in part to the variety of operational modes contained in the design of the processor I/O hardware. The proper operation of array resources such as the global flags and distributed handshaking signals, data and tag communication, and testbed-to-PC communication was tested by custom diagnostic routines.

ALGORITHM MAPPING AND TESTING

Sequential Model

The design approach used to map algorithms onto the HiSSAP testbed is shown in figure 4. Following the initial description of the application, a data simulator and one or more descriptions of the algorithm solutions are constructed. The data simulator provides an analytical data baseline with which to test the performance of each proposed algorithm model. The high-level description of the algorithmic approach is translated into a sequential model using a traditional programming language. A series of input data sets are constructed in the data simulator and processed by this sequential algorithm model to establish a performance baseline to which the parallel and systolic program implementations will be compared. The construction of a sequential mapping model may include optimizations that address such topics as numerical stability or computational efficiency. Otherwise, this programming is fairly straightforward.

PC-MATLAB[3] was selected as the software tool for all HiSSAP algorithm modeling because of its ability to manipulate array data and rapidly generate graphical displays.

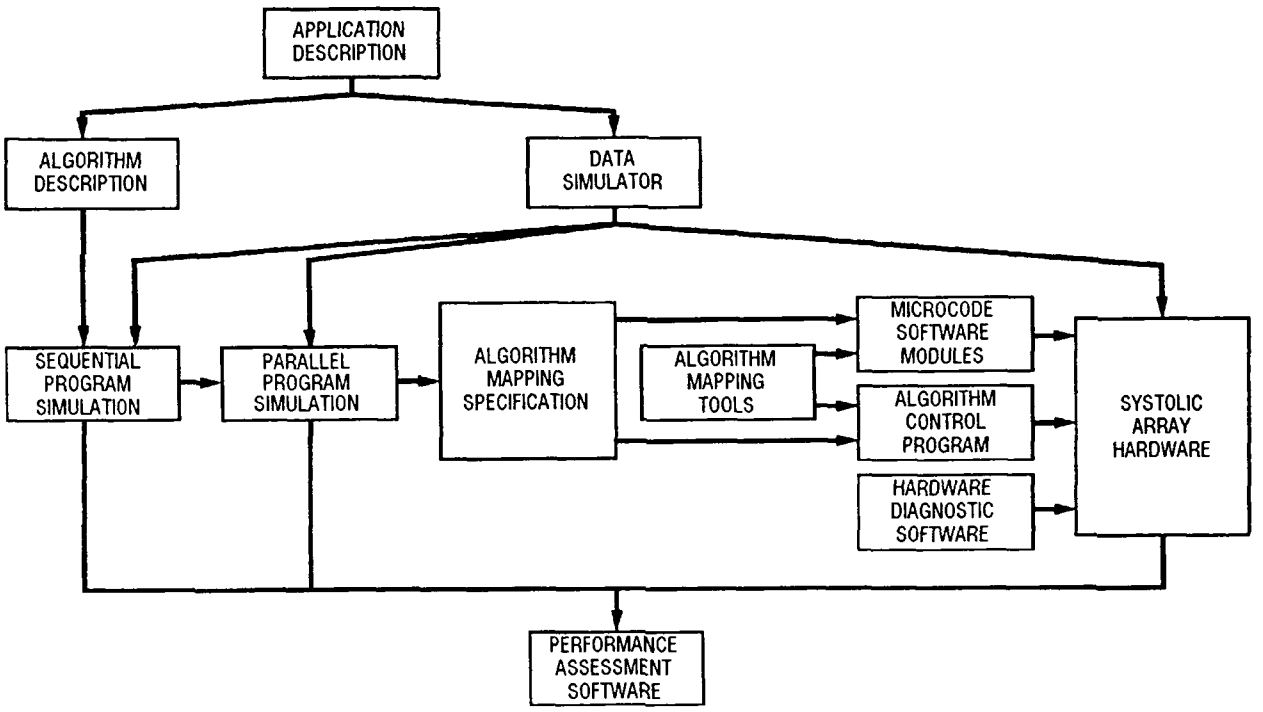


Figure 4. Algorithm mapping procedure.

Parallel Model

Once the sequential mapping has been baselined, a parallel model is then constructed. This translation requires a careful analysis of the structure of the algorithm to identify the computations that may be performed in parallel. The most difficult aspect of constructing this parallel model is assigning the proper grouping of computations and operand access at each computational node. No mapping aids existed that could automate this activity or evaluate the success achieved in efficiently using the resources of systolic architectures. The parallel program model of the multiple-signal classification (MUSIC) algorithm to be hosted on the systolic testbed was quite difficult to generate and debug using a general-purpose programming environment such as PC-MATLAB.[4] The development of specialized software tools targeted at this phase of application mapping is crucial to transitioning systolic technology into general usage.

[3] PC-MATLAB is a trademark of Mathworks, Inc.

[4] Dr. S. I. Chou at Naval Ocean Systems Center (NOSC), San Diego, formulated the MUSIC algorithm implementation and specified its parallel partitioning for use on the HiSSAP for the High Frequency Direction Finding Project.

Following the construction of the parallel model, computational equivalence to the sequential model was verified using the original baseline data sets.

An algorithm mapping specification is extracted from the parallel model prior to porting the algorithm onto the systolic testbed. The specification identifies the correspondence of computational tasks within the model with individual array element hosts. Also identified in the specification are those portions of the algorithm that are global in nature and must reside on the controller PC.

Several custom language tools were developed to facilitate the translation of the mapping specification into executable code on the systolic testbed (reference 11). Those portions of the application algorithm to be hosted on the array elements were translated into microcode modules to be loaded on the systolic testbed. The remaining portions were translated into an algorithm control program to be executed by the controller PC.

A final debugging and verification phase was conducted on the algorithm hosted on the testbed. A custom set of algorithm debugging tools was designed to provide user access to program and memory variables within the testbed hardware similar to that available in PC-MATLAB. Functional equivalence was verified with a custom set of algorithm debugging tools designed to provide user access to program and memory variables within the testbed hardware. This facility (references 8 and 11) used a system of set-scan diagnostic/control ports and clocking control on the testbed hardware combined with display screen and disk file management utilities residing on the personal computer controller.

LESSONS LEARNED

Although the HiSSAP testbed incorporated many internal hardware debugging features that were originally intended to support the initial functional testing of the system, the algorithm mapping effort also benefited from their inclusion. Once the algorithm code has been created and debugged on the systolic host processor, a greatly reduced set of architectural resources should be needed to support the run-time version of the application software on an embedded processor. This suggests that the initial mapping and checkout of program modules may best be undertaken on a development host that provides the user with an expanded diagnostic capability. Once the application coding has passed functional testing, it would be included in a software library available for hosting on production host systems. These systems are behaviorally equivalent to the development host systems, but their designs are simpler and less costly.

HARDWARE

Error Handling

The testbed hardware incorporated several circuits to monitor hardware fault conditions. Program memory parity errors and program stack overflow conditions were continuously monitored on each of the processor elements of the array. User programmable execution breakpoints and PC host trap handling of Institute of Electrical and Electronic Engineers (IEEE) exception conditions for numeric computations were also provided to aid in debugging the execution of the algorithms hosted on the testbed. The system was designed to report this information to the PC host computer and halt execution on the testbed at the offending instruction. This "report back" system was implemented as a wired or global signal to minimize the complexity of the systolic backplane. Due to the disparity in execution speeds of the HiSSAP testbed and the PC host, the fault status processing was handled in the system control module of the testbed system. The user could use the HiSSAP diagnostic utilities to interrogate the array elements, extract the source of the fault, and determine the course of action.

The implementation of these error and status report-back mechanism added significant complexity to the testbed hardware, but was absolutely necessary to verify proper system operation during all phases of the HiSSAP development. It is highly desirable to minimize this complexity to achieve a desirable cost/performance factor for the parallel processor used in embedded applications. The existence of powerful programming tools and debugging aids that rely heavily on these diagnostic hardware mechanisms factor into the cost of algorithm development. The following section describes a system implementation approach that addresses both of these considerations.

Development System. The development system should include two capabilities: programmable fault processing and fault status access.

Programmable Fault Processing. During system development, the response to execution events normally interpreted as faults may be primarily tailored for diagnostic purposes. It is often helpful during the integration phase of the hardware development to partition the reported system faults into categories. The user should have the capability to enable only the faults and condition flags deemed relevant at any stage of hardware or software development. This capability should be extended to the processor level for addressing those occasions when multiple-error conditions are simultaneously detected.

The user should also be able to assign the level of severity (and associated automatic hardware response) to each anticipated error reported by the processor system. An interruption of algorithm execution may not be appropriate for each system fault reported. As observed with the IEEE exception codes produced by the floating-point devices in the HiSSAP testbed, not all reported conditions indicate a failure of the current numerical computation. The user must be given the opportunity to configure the hardware to recognize a selected subset of possible errors as system hardware traps. The user should be allowed to direct the storage and subsequent handling of less urgent error (status) reports to specific diagnostic software modules.

Fault Status Access. The location and description of the condition (i.e., processing element number, fault, or condition type) should be automatically determined by the software development tools. It is important to minimize disruptions during software development by insulating the user from the tedium of manually evoking low-level diagnostic routines to interrogate and interpret the state of internal hardware error status registers.

Production System. The production system should include two capabilities: a fault hierarchy and fault tolerance.

Fault Hierarchy. A hierarchy of error handling should be adopted that routes the detected fault condition based on its severity. The nature of the error may dictate a minor variation of the programmed execution at the element. Other errors may require global coordination with other processor elements or user interaction to resolve.

Fault Tolerance. The effect of system errors caused by hardware failures within one or more array elements may be minimized by automatically instigating fault-tolerant reconfiguration of either system software execution or hardware data connectivity.

Architectural Implications of Error Handling. The issue of error handling is compounded if the subsystems being monitored are pipelined structures. The offending event may be difficult to locate, and the steps to reset the program to the pre-error condition may be difficult to ascertain. It is highly desirable to avoid pipeline structures if an instruction-by-instruction error-handling feature is to be implemented. An

alternate method that simplifies the task of properly handling system errors focuses on the structure of the application program. By restricting the processing of errors to regions interstitial to major sections of program code (i.e., the computational pipelines have been emptied or all pending I/O operations are complete) interference with the internal flow of execution can be minimized. This error-handling approach reduces implementation complexity at the expense of response latency. Depending on throughput requirements, this error-handling latency may not be compatible with the execution of real-time signal processing algorithms.

Processing Memory Requirements

The HiSSAP design was finalized before an accurate assessment of element memory requirements could be completed. Due to memory limitations within the testbed, the mappings of the MUSIC and finite impulse response (FIR) filter algorithms were less than optimum. The FIR filter was implemented as a cascaded system of desampling filter sections due in part to the limited amount of on-board memory (reference 12). The dimension of the spectral response produced by the MUSIC algorithm was also limited by the memory available on the processor elements.

Element Architecture

The architecture of the processing element was optimized for maximum throughput by dedicating separate cache memory to the floating-point ALU and multiplier devices. This minimized the latency in accessing operands for each processing chip but proved awkward when cascading mixed operations. The multiply-accumulate operation, so prevalent in signal processing applications, required additional clock cycles to complete on the HiSSAP processor due to the need for routing the intermediate results through memory.

The computational resources of any proposed array element should reflect the anticipated needs of the planned algorithm applications. Priority should be placed on minimizing the participation of local memory during chained computations. Two major benefits are realized by clever resource connectivity. The execution time of most chained operations can be greatly reduced and valuable memory space assigned to hold intermediate results may be released for other purposes. However, during the algorithm development phase, it may be advantageous to store the intermediate values of such computations in memory to accommodate user interrogation. This added bus and memory allocation would increase the complexity of the development system, but may be warranted to ease debugging of complex algorithms.

Address Generation

The HiSSAP processing elements contained a rudimentary address generation circuit that provided the capability for generating data memory address pointers. This feature enabled the programmer to generate pointers spanning large blocks of locally stored data while using a single instruction call. This approach greatly reduced the size requirements of program memory at each processing element.

SOFTWARE

Conceived as a research tool, the HiSSAP processor system was purposefully augmented with an extensive collection of data movement resources. This emphasis in system design provided the necessary flexibility to study the interaction of a broad range of algorithms on a multiprocessor host. The complexity of the software diagnostics developed for the HiSSAP reflected the general-purpose approach of the processor hardware architecture. For specialized or embedded applications, it is advantageous to resolve the architectural issues early in the system design phase and define a minimal set of programmable resources. One major goal of any system design of a parallel architecture is the efficient use of hardware computational resources. The major task in the mapping of algorithms onto parallel architectures deals with the management of data movement interior to the processing element and throughout the structure of the multiprocessor hardware. It is here, also, that the complexity in the diagnostic tools is the greatest. An effort to streamline or consolidate data movement resources while maintaining high computational throughput will result in significant reductions in the complexity of the diagnostic tools.

Performance of the algorithms hosted on the parallel processing host is difficult to quantify (reference 10). Our efforts on this topic were limited due to project time and budget but did highlight significant factors that should be addressed. Described below is a brief listing of capabilities that should be incorporated into tools developed for benchmarking parallel implementations.

Numerical Precision

Both the intermediate and final results of an algorithm may vary greatly due to the numerical precision used in the computations. In extreme cases, unstable or inconsistent results may occur. The word length of stored variables and the implementation of the fundamental mathematical operators are the primary system design factors that affect computational precision. To maintain numeric stability during the execution of an iterative computation loop, the required precision of the internal variables may greatly exceed that required at the I/O boundary of the algorithm. A software simulator is an effective tool to estimate the stability of algorithms that must operate on

hardware hosts with limited numerical precision. It is desirable to provide a means for adjusting the numerical precision of algorithm modeling software to accurately match that of the target hardware host. This ability to configure the model should extend to the manner to which the exception and error conditions are handled. By allowing the user to vary such parameters as numeric precision of modeled computations and intermediate variables, and the type of rounding or truncation, the simulator can be used to identify those portions of an algorithm requiring the greatest numeric precision.

It may be possible to relax the internal precision requirements an algorithm places on the host by reordering the sequence of mathematical operations used. Alternative algorithms may also be substituted that are more tolerant of ill-conditioned input data and maintain the condition number of internal matrix data at levels approximating that of the input data. The ultimate goal in either case is to achieve stable algorithm operation while producing output data of the required numerical precision.

Computational Efficiency

Several parallel architectural designs may appear to be suitable for hosting a desired algorithm. Peak computational throughput of a processor assumes the computations are performed at every available clock interval without regard to the issue of I/O data movement latency. This peak rating can be particularly misleading when applied to an array of processing elements. Simply multiplying the throughput figure of an individual processing element by the number of nodes in the array ignores the impact of access delays in a distributed memory architecture. Such factors as computational interdependence, approach to distributed data mapping, and bandwidth of interprocessor communication all contribute to the data latency at each computational element. This latency manifests itself as degraded computational throughput that may be a small fraction of the peak rating. Processor architectures using comparable computational resources may exhibit strikingly divergent throughput figures. Conversely, a single architecture may also exhibit great variation in throughput when hosting dissimilar tasks.

Communication/Computation Balance

The complexity of an architecture is greatly influenced by the structure and connectivity of the interprocessor data pathways. To reduce cost and complexity, connectivity among processors nodes within the array should be minimized. To maximize the computational throughput (by reducing the latency in accessing data in a distributed memory system) and minimize programming complexity, the architecture chosen must maximize connectivity. A compromise solution must be formulated that minimizes the latency while providing a realizable system architecture. A tool capable of modeling or measuring pathway usage of the array processor hosting specific algorithms would provide valuable guidance in the selection and evaluation of candidate architectures.

Identifying a system-wide optimum relationship between interprocessor communication bandwidth and intraprocessor computational loading is difficult. Several factors affect the definition of the problem. Fundamental to the discussion is the granularity of the programming model used to implement the algorithm. A fine-grain implementation maps the computations of the algorithm onto a large number of processing elements. Results of internal computations are usually passed to other processors in the array on a fairly regular basis. This implies a need for high-bandwidth interprocessor communication channels capable of supplying new operands at a rate approaching the computation rate of the element.

Some algorithms do not map well onto fine-grain computational hosts. A second class of distributed processing uses large groupings of complex numerical computations at each processing element. During the computations, all the required operands are available within the local memory of each element. This granularity of mapping often results in a reduced bandwidth demand on array interprocessor communication channels and a reduced average data access latency.

CONCLUSIONS

This document has described the lessons learned from the HiSSAP testbed development and HFDF system integration efforts. Clearly, the interaction of algorithm and architecture within a parallel processing host is complex. The topics described above serve mainly to identify major architectural design issues that must be addressed if optimum system performance is sought. Once the specific requirements of the hosted algorithm (or class of algorithms) are identified, a parallel processor architecture that uses its resources efficiently can be designed.

While the systolic mapping of the direction-finding application represents a significant accomplishment, the hardware and software diagnostic techniques developed for this effort have a more universal application.

REFERENCES

1. Speiser, J. M. 1986. "Linear Algebra Algorithms for Matrix-Based Signal Processing," Highly Parallel Signal Processing Architectures, SPIE Critical Review of Technology, Los Angeles, CA, SPIE, vol. 614-01.

2. Speiser, J. M. and H. J. Whitehouse. 1981. "Parallel Processing Algorithms and Architectures for Real Time Signal Processing," *Proceedings of the SPIE International Technical Symposium*, Real Time Signal Processing IV, vol. 298-01. 25-28 Aug 1981, San Diego, CA.

3. Kung, H. T. 1982. "Why Systolic Architectures?," *Computer, IEEE Society*, vol. 15, no. 1.

4. Symanski, J. J. 1983. "Implementation of Matrix Operations on the Two-Dimensional Systolic Array Testbed," *Proceedings of the SPIE International Technical Symposium*, Real Time Signal Processing, vol. 431-19. 21-26 Aug 1983, San Diego, CA.

5. Drake, B. L., F. T. Luk, J. M. Speiser, and J. J. Symanski. 1987. "SLAPP: A Systolic Linear Algebra Parallel Processor," *IEEE Computer*, vol. 20, no. 7, pp. 45-49.

6. Symanski, J. J., T. Henderson, J. Shirasago, J. Celto, and B. Drake. 1987. "SLAPP: A Special Purpose Multiprocessor Array for Signal Processing and Linear Algebra," *Proceedings of the SPIE International Technical Symposium*, Advanced Algorithms and Architectures for Signal Processing II, vol. 826-32. 18-19 Aug 1987, San Diego, CA.

7. Symanski, J. J., K. Bromley, and T. B. Henderson. 1988. "The Video Analysis Transputer Array (VATA)," *Proceedings of the SPIE 32nd Annual International Technical Symposium on Optical and Optoelectronic Applied Science and Engineering.*

8. Loughlin, J. P. 1987. "NOSC Advanced Systolic Array Processor (ASAP)," *Proceedings of the SPIE International Technical Symposium*, Real Time Signal Processing X, vol. 827-13. 18-19 Aug 1987, San Diego, CA.

9. Loughlin, J. P. 1989. "Multiple Signal Classification (MUSIC) Algorithm Hosted on the High Speed Systolic Array Processor (HiSSAP)," *Proceedings of the SPIE International Technical Symposium*, Real Time Signal Processing XI, vol. 977.

10. Loughlin, J. P. and F. M. Tirpak, Jr. 1990. "Systolic Signal Processor/High Frequency Direction Finding, Final Test Report," NOSC TR 1369 (Oct). Naval Ocean Systems Center, San Diego, CA.

11. Tirpak, F. M. Jr. "High Speed Systolic Array Processor (HiSSAP) Software Development Environment, Lessons Learned," NOSC technical report to be published.

12. Tirpak, F. M. Jr. "Design and Implementation of a Multistage Narrowband FIR Filter on the High Speed Systolic Array Processor (HiSSAP)," NOSC technical report to be published.

APPENDIX A

SYSTOLIC ARCHITECTURES

EVOLUTION OF PROCESSING ARCHITECTURES

Historically, the enhancement of computational throughput in signal processing hosts has been achieved through advancements in process technology. Both speed and device density have shown a steady increase over the past two decades with a corresponding increase in system performance. The device technology is reaching a maturity that may limit its ability to allow traditional uniprocessor designs to provide the next increment of computational throughput required by the modern signal processing algorithms now emerging. Many system architectures that contain multiple computational engines operating in parallel have been proposed to alleviate the bottleneck existing in uniprocessor designs. This concept, while not new, has enjoyed limited application due in large part to the difficulty in programming a device of this type. The first parallel architectures consisted of multiple processing elements connected together on a shared bus and all vying for access to a single shared memory. The bottleneck caused by the bus bandwidth limitations of such a topology has provided the motivation to study other forms of parallel architectures.

SYSTOLIC ARCHITECTURE

A parallel processor architecture that uses a single communication bus and shared memory does not appear to provide the system designer the freedom to scale the throughput of the machine to match the computational requirements. A systolic array is an alternative architecture that uses a multiple bus structure and distributed memory to address the limitations existing in the previous approach. The term "systolic" as it refers to computer architectures encompasses a broad spectrum of hardware topologies. Many multiprocessor structures (hypercube, connection machine, iWarp, Touchstone[5]) use differing interprocessor communication approaches and nodal functional capabilities. The only shared characteristic that bind these diverse structures under the common category of systolic is the enhanced computational throughput achieved by the use of multiple processing nodes in a distributed bussing system.

PERFORMANCE CRITERIA

The enhanced throughput achievable with the systolic approach is based on two critical factors. First, the computational throughput of a system can be enhanced by

[5] Touchstone is a trademark of Intel Corporation.

using multiple processors working concurrently on small portions of a problem. Second, no additional interprocessor communication overhead is incurred as the size of the processor array varies. If both of these criteria are satisfied, incremental expansion in the number of processing nodes should be reflected by a corresponding overall increase in computational throughput. The degree to which this enhanced throughput is achieved depends not only on the hardware implementation, but the interaction of this architecture with the specific algorithm being hosted. It is commonplace to find computers rated in peak computational throughput, often in terms of instructions per second (IPS) or floating-point operations per second (FLOPS). The actual computational rate may be significantly less than the peak computational rate due to inefficient use of resources. Separate treatments of the hardware and software factors affecting computational throughput in a systolic architecture will be presented here and can be found in references A-1 and A-2.

ARCHITECTURAL CHOICES

Massive computational throughputs can be achieved using any of several possible systolic topologies. Depending on the task to be hosted, the processor system may take the form of a linear, two-dimensional, or n-dimensional topology. To determine the appropriate architecture for a given algorithm, several key factors in mapping must be evaluated. Paramount in the efficient use of such a parallel structure is the degree of parallelism in the algorithm itself. In addition to the identification of operations that can be structured to be performed simultaneously, the balance of computational loading and interprocessor communication capabilities must be carefully considered. In a uniprocessor architecture, all the data associated with the computational task resides in a single memory. The delay in retrieving any required data is universal to any operation performed by that single processor. This approach to data retrieval works well when the central processing unit (CPU) has instruction rates comparable to the access times of the memory. As additional processors are added to the array to accommodate the need for greater computational throughput, a corresponding increase in data storage access bandwidth is required. In addition to the memory bandwidth, the overhead task of managing the usage of this shared resource becomes impractical. The systolic topology, when properly applied, provides a favorable balance between individual CPU operations and communication of operands from neighboring segments of distributed memory. This distribution of memory throughout the processor array also eliminates the need for complex global memory management schemes.

While the systolic architecture promises to offer a scalable solution to the ever increasing computational throughput demands of modern signal processing algorithms, it is difficult to obtain peak performance for all applications. The apparent simplicity of this parallel architecture obscures the underlying complexity encountered in restructuring sequential algorithms to efficiently use the parallel resources of this architecture.

Perhaps the most important factor in the successful application of systolic arrays to signal processing is processor utilization efficiency. In the context of this report, this efficiency refers to that fraction of the array's computational resources used during each instruction cycle. As discussed later, while the utilization efficiency is measured in terms of CPU computational throughput, underlying this performance is a carefully matched mechanism for interprocessor data transfer throughout the processor array. The systolic array architecture provides the system designer with a latitude to match the interprocessor communication performance to the requirements of many diverse classes of algorithms. While total utilization is rarely achieved in any parallel processing architecture, systolic arrays provide scalable solutions that may be optimized for general classes of computations. Achieving the performance potential of systolic arrays through algorithm/architecture tuning precludes its use as a general purpose computational host. It is best suited for embedded front end processors that provide the specialized computational resources for a chosen class of matrix-based or vector-based algorithms.

GENERAL CLASSIFICATIONS OF ARRAYS

The simplest model of a systolic array consists of an array of identical processing elements executing the same string of instructions in unison. The instruction commands are often designed to emanate from one central control point. While limited data-dependent execution may occur at each processing node, no variation in system program execution is allowed at the global level. Data movement within the array is defined only between neighboring processing elements. Program execution is linked with data movement among the elements of the array, thereby guaranteeing the proper staging of operands prior to each computation. This description of systolic operation is referred to as single-instruction multiple-data (SIMD). Algorithms that can be mapped onto a systolic processor array operating in a SIMD mode must not require global data communication other than that of initialization. Certain highly structured algorithms with an intrinsic balance between computational load and operand movement allow complete utilization of system resources in a systolic architecture. Arbitrary length vector-vector inner products can be computed on a one-dimensional systolic array of corresponding length. Using a two-dimensional mesh-connected systolic array of processors, vector-vector outer products, vector-matrix, and matrix-matrix multiples can also attain complete resource utilization efficiency. Finite impulse response (FIR) filters, and fast Fourier transforms (FFTs) are examples of signal processing algorithms that may be structured in SIMD form and require minimal global control.

A more complex systolic model uses an array of processors in which each processor may be called upon to execute different tasks. This structure is more commonly referred to as a multiple-instruction multiple-data (MIMD) processor. The computational requirements of a broader range of algorithms can be addressed by this

structure at the cost of increased software complexity. Several new mapping issues must be addressed to successfully coordinate the execution of multiple tasks within the processor array. The regular data flow and simplicity of unified program execution typified by a SIMD machine is replaced by a complex interprocessor synchronization scheme. Due to variations in computational loading assigned to each processor element and the data interdependencies among elements in the array, the appearance of idle resources is difficult to avoid. Compounding the problem is the expansive collection of data transfer topologies. The nearest neighbor data transfer convention described above must be augmented with such pathways as row or column broadcasting to achieve maximum use of the data buses.

TASK PARTITIONING (SYSTEM LEVEL)

Several factors bear on the selection of the type of array structure (or structures) best suited to a particular algorithm. A signal processing task may be composed of multiple subtasks with differing processing requirements. Two alternative system approaches to accommodating these processing requirements maximize the overall data throughput by different means. One possible solution, shown in figure A-1, uses a system of multiple daisy-chained systolic architectures, the structure of each reflecting a minimal computational and communication approach for each subtask. As processing is completed on each subtask, the intermediate data is passed to the next systolic member of the processor chain. The major advantage of this approach is the high degree of computational efficiency achieved within each stage of the algorithm. The design of each subarray processor is optimized for a specific subtask, which should yield simplified structure. The overhead associated with the communicating of intermediate data between the subarray units at the completion of each subtask may seriously undermine the aggregate system throughput. This overhead is related to the size of the minimum block of input data necessary to support valid computation within processing subarrays and the channel bandwidth.

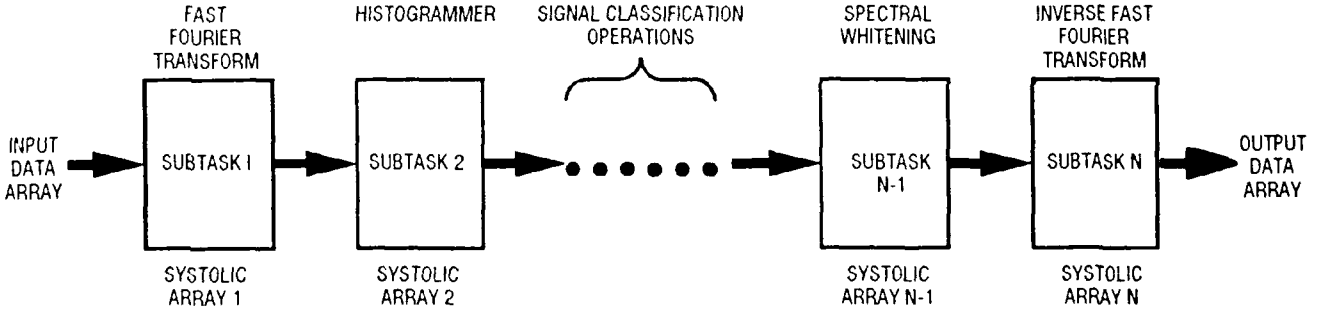


Figure A-1. Subtask-partitioned algorithm mapping.

Another equally legitimate solution to this multitask algorithm is processing "in place" using a single general-purpose array architecture. In this approach, the computational and communication requirements of all of the subtasks are considered as a whole in the design of this unified parallel processor host. This generally results in a single system design complexity greater than any one of the component subarray architectures of the daisy-chained approach. This general-purpose design does offer several advantages that offset the design complexity issue. A single system architecture reduces the complexity of the overall design and requires one set of programming tools. The data movement delays (described above) may be totally eliminated.

COMMUNICATION STRUCTURES

Movement of data throughout the processor array is one of the most complicated issues in the mapping of algorithms onto the systolic architecture. In a sequential implementation, the programmer is responsible for the order in which the computations are performed, therefore the existence of the required operands for each successive computation can be guaranteed by the inherent structure of the program. Parallel implementation of algorithms introduces an additional execution concern. Due to the physical partitioning of system memory into local caches for each processor, program access to any arbitrary variable must include reference to physical as well as logical location. The manner in which the data is distributed across the element memories and the availability of the communication bus(s) required to gather the operands to the site of each computation greatly impact system throughput.

SYNCHRONIZATION

As described earlier, an important characteristic of the simplest model of systolic processing (SIMD) is identical and synchronized program execution within the elements of the array hardware. This lock-step execution of program code is typically accomplished in either of two ways: (1) single-source instruction distribution or (2) element program synchronization using semaphores. The first technique employs a common program sequencer to control the computations throughout the array. The alternate approach locks the program execution of all of the otherwise autonomous processing elements by using a global handshaking protocol. Any differences in the parallel processing activity in either array implementation are indistinguishable. As noted earlier, the control of a systolic array processor operated in this manner is trivial, but is effective in hosting a special class of algorithm.

This execution mechanism must be expanded to support the hosting of more advanced signal processing algorithms such as the multiple signal classification (MUSIC) algorithm. This expanded model must support the coordinated execution of heterogeneous program tasks on the processing elements within the array. This disparity of tasks

assigned to processing elements usually causes an imbalance of execution times. The specification of data movement within the array becomes much more complex and requires an extension to the concept of program execution synchronization. Several possible synchronization schemes exist to manage the coordinated movement of data among heterogeneous program nodes in the array. Among the most widely used are (1) padding of execution time, (2) transfer handshaking, and (3) input/output buffering.

Program Padding

The first technique achieves synchronized operation by inserting "no op" instructions into the program code of selected processing elements. By adding the appropriate number of these instructions into the execution path of all but the slowest program modules, time alignment of key computations occurring within each of the processing elements can be maintained. While incurring no changes to the system hardware, this technique is implemented at the cost of increased complexity in the software generation tools. An additional penalty is the reduced computational efficiency caused by periodically idling processors during the algorithm execution. The use of "no op" instructions is a crude method of inhibiting the initiation of each code segment until all the prerequisite computations and data transfers within the array are complete. It guarantees the integrity of the algorithm.

Transfer Handshaking

A closer study of this synchronization issue as it applies to systolic processors indicates that a hardware-based solution to this problem is possible using another mechanism. This alternate approach avoids the complexity associated with the "no-op" style compiler by using hardware handshaking protocols. This type of resynchronization is commonly implemented by a system of interprocessor handshaking. This approach uses "do loop" style interrogation for the required handshaking signal to arrive at each processor. This technique can be used in multidimensional array architectures to effect global or regional synchronization as the algorithm dictates. Since synchronism is established at algorithm run time by the system hardware, no additional complexity is incurred in the algorithm mapping tools. To prepare for a transfer of data, each processing element must include a brief conditional test of the handshaking status. Successful coordination of all the processing elements within the array can be achieved without the need for "clock counting." The penalty is similar to the first option, loss in processor efficiency due to idle states occurring during handshake processing and the associated wait.

Input/Output Buffering

Input/output data buffering is a third alternative for integrating the execution of heterogeneous software modules. This approach entails the decoupling of data transfer

and computation functions at each element. This is usually done by using a pair of dedicated program processors within each element. The first processor is responsible for performing numeric computations, while the task of communicating with neighboring elements is relegated to an auxiliary processor. With each array node implemented as a pair of processors, the functions of computation and communication can be decoupled and more freedom can be exercised to optimize system efficiency. The wait states previously used to synchronize processors need only affect the communications processor of each element. Several tasks may be queued within a processing element and be evoked only when the corresponding operands or resultants have been staged within the cache memory. With greatly relaxed constraints on the relative timing associated with computations within neighboring processor, a more powerful communication network can be supported.

By using an accompanying identity tag with the buffered data approach, the destination of each data element can be routed throughout the array via the intelligent action of the communications network. The physical site of "nearest neighbor" computations may be extended to include any processor within the array. In an array using tag information sent with each data word, the focus of the algorithm mapping tools may be directed more at the computations and less at the movement of data. Correctly implemented, these tools will provide a means of rapidly mapping a broad spectrum of algorithms while maintaining high processor utilization.

COMPUTATION/COMMUNICATION BALANCE

An algorithm that requires a large number of data transfers for each numerical computation performed is described as "I/O bound." Many image processing tasks fall into this category being composed of many relatively primitive computations linked by massive pixel movement operations. The execution of such algorithms on systolic processors can be quite efficient while data bottlenecks on the shared communication networks in traditional array processors renders much of their peak computational throughput unavailable.

Another category termed "compute bound" refers to those algorithms that perform complex (time consuming) operations on moderate groupings of data, often resulting in an output data structure of substantially smaller dimension. An array processor optimized for this type of algorithm places fewer demands on the communication mechanism. While the bandwidth requirement can be substantially lower, often a more sophisticated data management approach is required.

ARRAY DATA BUS IMPLEMENTATIONS

For maximum data transfer bandwidth between processing elements, word-wide parallel transfers are optimum. If each processing element is required to support

several communication ports, such a system implementation can become quite costly in several aspects. The power needed to operate a multitude of output driver/receivers can become prohibitive, especially if monolithic integration is considered as an evolutionary goal of the system design. In addition, the reliability and manufacturability of a systolic backplane is adversely affected by such complex parallel bus implementations. It is, therefore, advantageous to trim the performance and complexity of the bus structure to the minimum level necessary to support efficient execution of the algorithm. Due to the number of I/O pins needed for this word-wide bus implementation, it appears unlikely that larger arrays with greater dimensional connectivity could be practical.

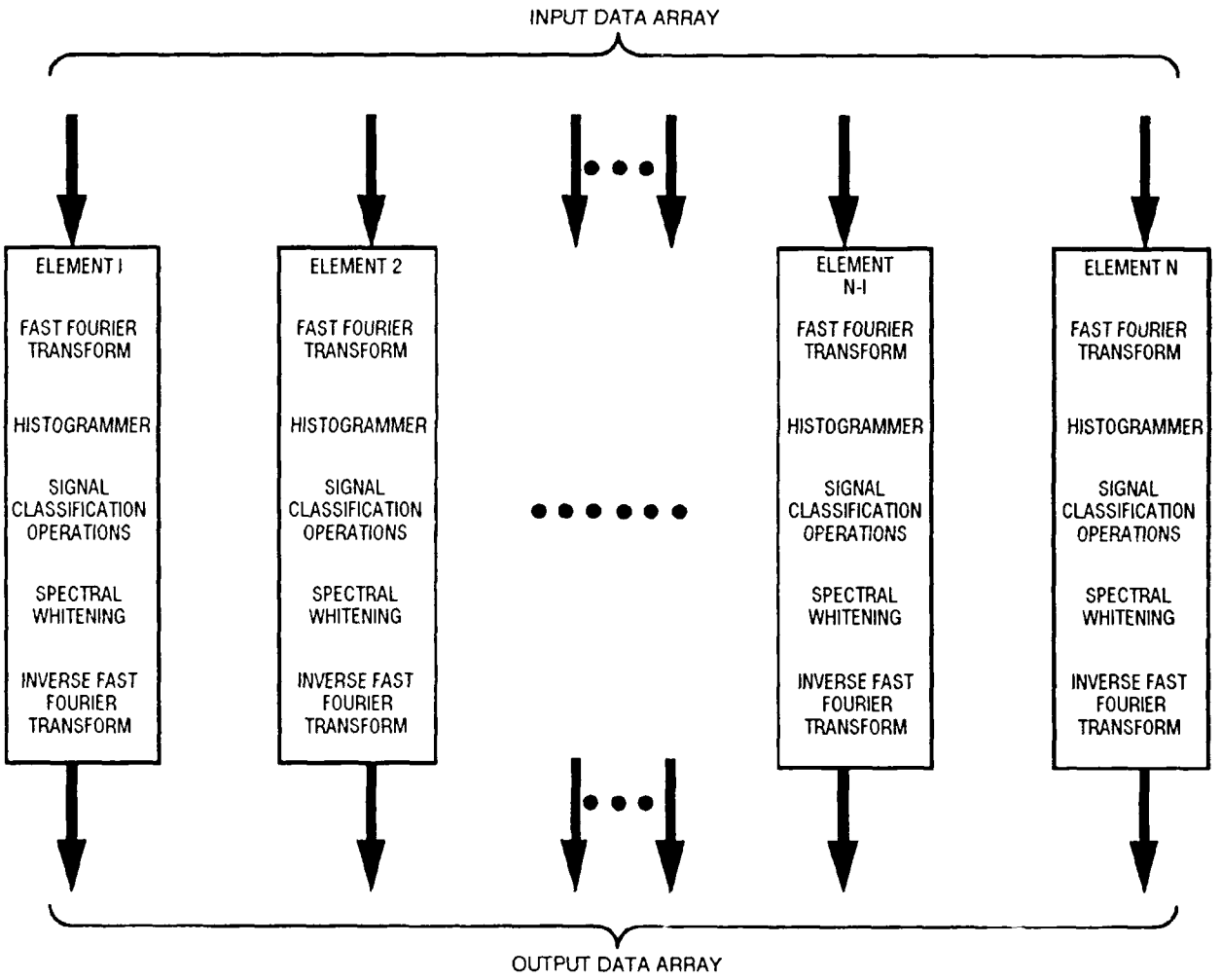


Figure A-2. "In place" algorithm mapping.

An alternative approach employs data transfers occurring over multiple clock cycles. Successive slices of a data word are transmitted over a narrower bus in a time-multiplexed manner. This reduces the complexity of the array interconnects at the expense of both the element's interface design complexity and bus occupancy time per transfer. Additional circuitry within each element necessary for the reconstruction of the parallel data word may be of minimal consequence, but the impact on bus bandwidth (inverse of the time/transfer) must be considered carefully with respect to the demands of the mapped algorithms. In this instance, when an algorithm may underuse the full bandwidth of a word-wide parallel bus structure, serializing the communication between processor elements into slices as small as 1 bit wide may be advantageous. The bandwidth and complexity of each communication channel can be scaled down without affecting the computational throughput.

Two-way data communication with neighboring processors can be addressed in either of two ways: a single bidirectional (full-duplex) channel, or paired unidirectional (half-duplex) channels. Each approach presents the system designer with a unique set of engineering advantages. Full duplex communication minimizes interconnection complexity at the backplane of the array. The savings in system complexity due to the reduced proliferation of interconnects can be substantial for multidimensional array configurations. The major drawback to effectively using a communication system constructed of bidirectional channels is the increase in complexity required at each computational element. To avoid collisions while moving data through this shared communication resource, transfer coordination must be established between processing elements.

The paired half-duplex channels approach supports two-way communications via two dedicated unidirectional data subchannels. The width of these subchannels are typically identical and differ only in assigned data communication direction. No bus arbitration is required and the bidirectional buffering of data in neighboring processors can be easily done without undue complexity in the interface circuitry.

HOST CONTROL

For embedded applications on the systolic array, no user interaction is generally necessary. However, to support algorithm mapping and debugging, it is necessary to provide user access to the contents of array internal memory. Systolic array architectures usually contain only the connectivity necessary to communicate data throughout the structure. User access to memory variables contained within the array can be supported via diagnostic I/O program code embedded into the algorithm code. This method is not trivial and impedes the rapid debugging of program code.

It is also desirable to provide a means for the user to easily down-load developmental program software and control its execution on the systolic hardware. It is desirable

to perform these operations in a manner that does not alter the execution of the target algorithm on the testbed.

SUMMARY

The comments presented above address design topics germane to specification of a parallel processing architecture. This appendix does not represent an exhaustive catalog of issues facing the design engineer. The ultimate priorities placed on these and other issues associated with parallel processor design and the mapping strategies used in mapping matrix-based algorithms will depend on the specific application(s) planned. The design of each systolic architecture should reflect the unique computational and communication requirements of the algorithm (or class of algorithms) to be hosted.

REFERENCES

A-1. Loughlin, J. P. 1987. "NOSC Advanced Systolic Array Processor (ASAP)," *Proceedings of the SPIE International Technical Symposium*, Real Time Signal Processing X, vol. 827–13. Aug 1987, San Diego, CA.

A-2. Tirpak, J. M. Jr., "High Speed Systolic Array Processor (HiSSAP) Software Development Environment, Lessons Learned," NOSC technical report to be published.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1 AGENCY USE ONLY *(Leave blank)*	2 REPORT DATE	3 REPORT TYPE AND DATES COVERED
	May 1991	Final: October 1983—October 1990

4 TITLE AND SUBTITLE
HIGH SPEED SYSTOLIC ARRAY PROCESSOR (HiSSAP) SYSTEM DEVELOPMENT SYNOPSIS: Lesson Learned

5 FUNDING NUMBERS
PE: NIF, 604507N
WU: DN308 022
PN: 76-EE3401

6 AUTHOR(S)
J. P. Loughlin

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Ocean Systems Center
San Diego, CA 92152–5000

8 PERFORMING ORGANIZATION REPORT NUMBER
NOSC TR 1432

9 SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Sea Systems Command
Washington, DC 20362

10 SPONSORING/MONITORING AGENCY REPORT NUMBER

11 SUPPLEMENTARY NOTES

12a DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b DISTRIBUTION CODE

13 ABSTRACT *(Maximum 200 words)*

This report documents the design rationale of the High Speed Systolic Array Processor (HiSSAP) testbed. In addition to reviewing general parallel processing topics, the impact of the HiSSAP testbed architecture on the top-level design of the diagnostic and software mapping tools is described. Based on the experience gained in the mapping of matrix-based algorithms on the testbed hardware, specific recommendations are presented in the form of "lessons learned," which are intended to offer guidance in the development of future Navy signal processing systems.

14 SUBJECT TERMS
High Speed Systolic Array Processor (HiSSAP)
parallel processing
signal processing

15 NUMBER OF PAGES
39

16 PRICE CODE

17 SECURITY CLASSIFICATION OF REPORT	18 SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20 LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
J. P. Loughlin	(619) 553-2541	Code 761

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0144	R. November	(1)
Code 76	F. M. Tirpak	(1)
Code 7605	Dr. K. Bromley	(1)
Code 761	Dr. G. Byram	(6)
Code 761	F. M. Tirpak,Jr.	(1)
Code 761	J. P. Loughlin	(10)
Code 952B	J. Puleo	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(3)

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

NOSC Liaison Office
Washington, DC 20363-5100 (1)

Center for Naval Analyses
Alexandria, VA 22302-0268 (1)

Naval Air Development Center
Warminster, PA 18974-5000 (3)